Learning Model Transformation Patterns using Graph Generalization

Hajer Saada, Marianne Huchard, Michel Liquière, Clémentine Nebut

Learning Model Transformation Patterns using Graph Generalization

Hajer Saada, Marianne Huchard, Michel Liquière, and Clémentine Nebut

LIRMM, Université de Montpellier 2 et CNRS, Montpellier, France,
first.last@lirmm.fr

Abstract. In Model Driven Engineering (MDE), a Model Transformation is a specialized program, often composed of a set of rules to transform models. The Model Transformation By Example (MTBE) approach aims to assist the developer by learning model transformations from source and target model examples.In a previous work, we proposed an approach which takes as input a fragmented source model and a target model, and produces a set of fragment pairs that presents the many-to-many matching links between the two models. In this paper, we propose to mine model transformation patterns (that can be later transformed in transformation rules) from the obtained matching links. We encode our models into labeled graphs that are then classified using the GRAAL approach to get meaningful common subgraphs. New transformation patterns are then found from the classification of the matching links based on their graph ends. We evaluate the feasibility of our approach on two representative small transformation examples.

1 Introduction

MDE is a subfield of software engineering that relies on models as a central artifact for the software development cycle. Models can be manually or automatically manipulated using model transformations. A model transformation is a program, often composed of a set of transformation rules, that takes as input a model and produces as output another model. The models conform to meta-models, as programs conform to the programming language grammar. If we would like to transform any Java program into any C++ program, we would express this transformation at the level of their grammars. In MDE, model transformations are similarly expressed in terms of the meta-models. Designing a model transformation is thus a delicate issue, because the developer has to master the specialized language in which the transformation is written, the meta-modeling activity, and the subtleties of the source and the target meta-models. In order to assist the developers, the MTBE approach follows the track of the "Programming By Example" approach [6] and proposes to use an initial set of transformation examples from which the model transformation is partly learnt. The first step of the MTBE approach consists in extracting matching links, from which the second step learns transformation rules. Several approaches [1,15,12] are proposed for the second step, but they derive element-to-element (one-to-one) rules

that mainly express how a source model element is transformed into a target model element. In this paper, we propose to learn transformation patterns of type fragment-to-fragment (many-to-many) using the output of a previous work [13] that consists in generating matching links between source and target model fragments. We encode our models and model fragments as labeled graphs. These graphs are classified through a lattice using a graph mining approach (GRAAL) to get meaningful common subgraphs. The matching links are then classified using Formal Concept Analysis, the lattice of source graphs and the lattice of target graphs. New transformation patterns are discovered from these classifications that can help the designer of the model transformation. We evaluate the feasibility of our approach on two representative transformation examples.

The next Section 2 gives an overview of our approach. Section 3 presents the transformation pattern mining approach and Section 4 evaluates its feasibility. Section 5 presents the related work, and we conclude in Section 6.

2 Approach Overview

In Model-Driven Engineering, model transformation are programs that transform an input source model into an output target model. A classical model transformation (`UML2REL`) transforms a UML model into a RELational model. Such transformation programs are often written with declarative or semi-declarative languages and composed of a set of transformation rules, defined at the meta-model level. The meta-model defines the concepts and the relations that are used (and instantiated) to compose the models. For example, the UML meta-model

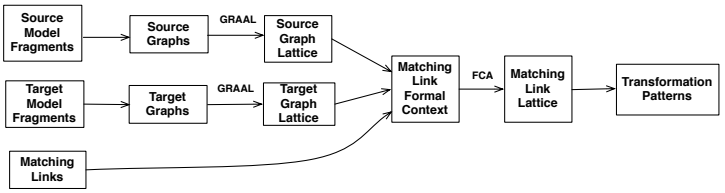


Fig. 1. Process overview

contains the concept of *Class* which owns *Attributes*. This can used to derive a UML model composed of a class *Person* owning the attribute *Name*. In the `UML2REL` example, a very simple transformation pattern would be: *a UML class owning an attribute is transformed into a relational table owning a column*. In this paper, our objective is to learn such transformation patterns that express that a pattern associating entities of the source meta-model (*e.g. a UML class owning an attribute*) is transformed into a pattern associating entities of the target meta-model (*e.g. a relational table owning a column*). Fig. 1 provides an overview of our process. Let us consider that we want to learn rules for

transforming UML models to relational models. Our input data (see Fig. 2) are composed of: fragmented source models (a UML source fragment is given in Fig. 3(a)); fragmented target models (a relational target fragment is given in Fig. 3(b)); and matching links between fragments established by experts or by an automatic matching technique. For example a matching link (`L1`) is established in Fig. 2 between the UML source fragment of Fig. 3(a) and the relational target fragment of Fig. 3(b).

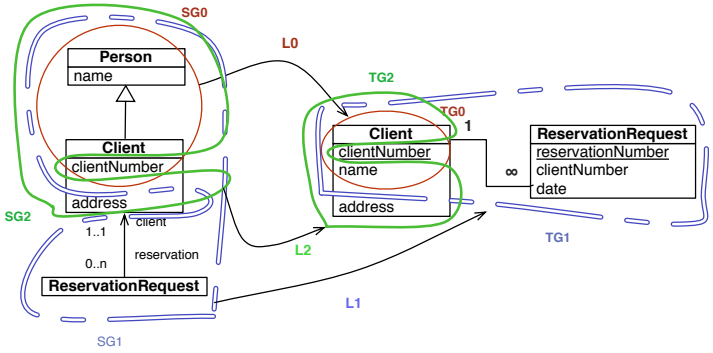
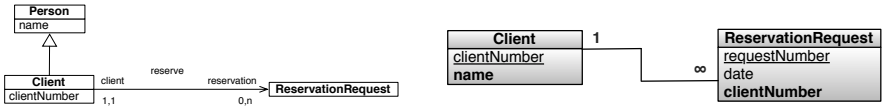


Fig. 2. Three matching links between fragmented UML and relational models (this figure is more readable in a coloured version)

Matching links established by experts or by automatic methods can be used to form a set of model transformation patterns. For example, the `L2` matching link gives rise to a transformation pattern which indicates that *a UML class (with an attribute) with its super-class (with an attribute) is transformed into a unique table with two columns, one being inherited*. Nevertheless, matching links often correspond to patterns that combine several simpler transformations or are triggered from domain knowledge. Besides, they may contain minor errors (such as a few additional or missing elements, for example, column `date` of Table `ReservationRequest` has in fact no equivalent in Class `ReservationRequest`). Moreover, what interests us is beyond the model domain. We do not want to learn that Class `Client` is transformed into Table `Client`, but rather that a UML class is usually transformed into a table.

Our output is composed of a set of model transformation patterns. Some can directly be inferred from initial matching links (as evoked previously), and some will be found thanks to graph generalization and matching link classification. From our simple example, we want to extract the model transformation pattern presented in Figure 4, whose premise and conclusion patterns do not appear as such in the initial set of matching links ($\hookrightarrow$ means "is transformed into").



(a) UML source fragment (b) Relational target fragment

Fig. 3. An example of UML and relational models

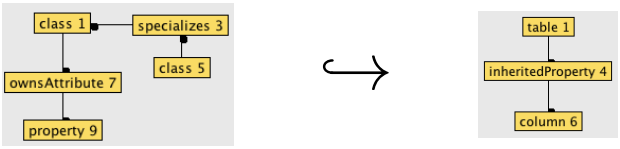


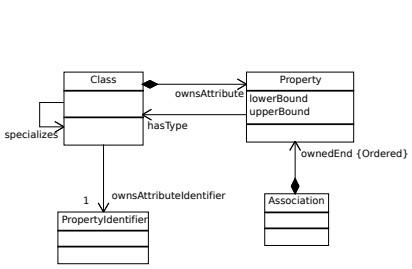
Fig. 4. Transformation pattern: a class specializing a class with an attribute (in UML model) is transformed into a table with an inherited column (in relational model).

3 Model Transformation Pattern Generation

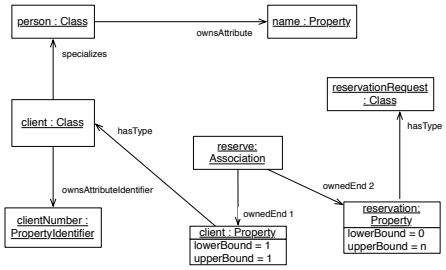
From model fragments to graphs For our example, the source meta-model is inspired by a tiny part of the UML metamodel (see Figure 5(a)), while the target meta-model has its roots in a simplified relational data-base meta-model (see Fig. 5(c)). The models often are represented in a *visual* syntax (as shown in Fig. 3(a) and Fig. 3(b)) for readability reasons. Here we use their representation as instance diagrams of their meta-model (using the UML instance diagram syntax). For example, the UML model of Fig. 3(a) is shown as an instantiation of its meta-model in Fig. 5(b), where each object (in a rectangular box) is described by its meta-class in the meta-model, valued attributes and roles conforming to the attributes and associations from the meta-model: *e.g.* `person` and `client` are explicit instances of `Class`; `client:Class` has a link towards `person` labeled by the role `specializes`; `client:Property` has the attribute `lowerBound` (1).

To extract expressive transformation patterns, we transform our models using their instance diagram syntax, into simpler graphs which have labeled vertices. We limited ourselves to locally injective labelled graphs. A locally injective graph is a labeled graph such that all vertices in the neighbor of a given vertex have different labels. This is not so restrictive in our case, because the fragments identified by the experts rarely include similar neighborhood for an entity. Here are the rules that we use in the transformation from simplified UML instance diagrams to labeled graphs. We associate a labeled node to Objects, Roles, Attributes, Attribute values. Instance diagram of Figure 5(b) and corresponding labeled graph from 6(a) are used to illustrate the transformation: `person:Class` object is transformed into node 1 labeled `class_1` and one of the attribute value `1` is transformed into node 13 labeled `one_13`. Edges come from the following situations: an object has an attribute; an attribute has a value; an object has

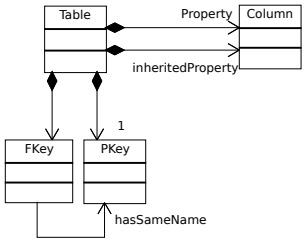
a role (is source of a role); an object is the target of a role. For example, for the property which has an attribute lowerBound (equal to `zero`), there is a corresponding edge `(property_17, lowerBound_18)`.



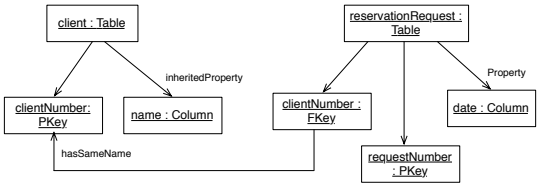
(a) UML metamodel



(b) UML model of Fig 3(a) in instance diagram syntax



(c) Relational metamodel

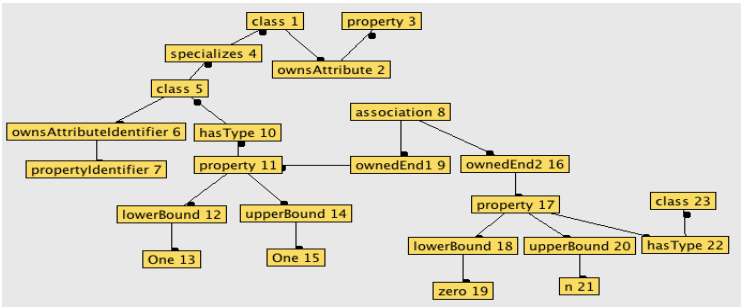


(d) Relational model of Fig 3(b) in instance diagram syntax

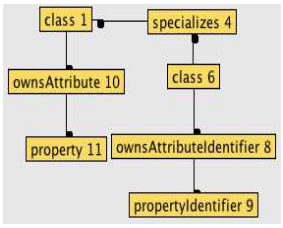
Fig. 5. Source/target metamodel and model, UML (upper par), relational (lower part)

Classification of graphs (GRAAL approach) After the previous step, we obtain a set of source graphs, and a set of target graphs. We illustrate the remainder of this section by using the three source graphs of Fig. 6, the three target graphs of Fig. 7, and the matching links (Source graph i, Target graph i), for i $\in \{0, 1, 2\}$. To get meaningful common subgraphs (on which new transformation patterns will be discovered), we use the graph mining approach proposed in [7] and its derived GRAAL tool. In this approach, examples are described by a description language L provided with two operations: an $\leq$ specialization operation and an $\otimes$ operation which builds the least general generalization of two descriptions. A generalization of the Norris algorithm [11] builds the Galois lattice. Several description languages are implemented in GRAAL, and especially a description based on locally injective graphs. $\otimes$ operation is the reduction of the tensor product of graphs, also called the Kronecker product [14]. We independently classify source graphs and target graphs. Classification of source graphs

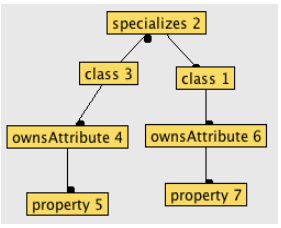
produces the lattice of Fig. 8(a). For example, in this lattice, Concept `sfc012` has for intent a subgraph of source graphs 0, 1 and 2 representing a class which specializes a class which owns an attribute. Classification of target graphs produces the lattice of Fig. 8(b). In this lattice, Concept `tfc012` has for intent a subgraph where a table has an inherited property.



(a) Source graph 1



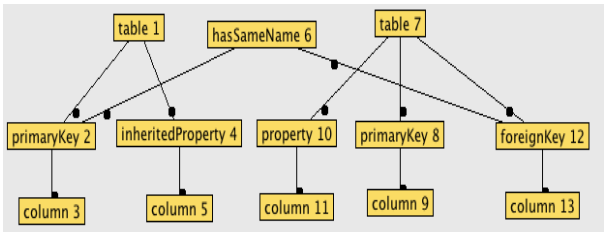
(b) Source graph 0



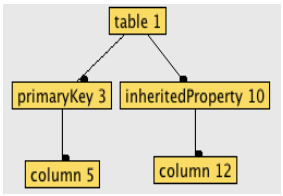
(c) Source graph 2

Fig. 6. Source graphs

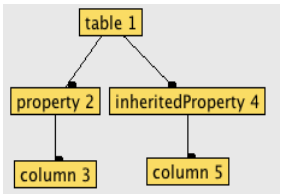
Classification of transformation links In the previous section, we have shown how Galois lattices can be computed on the labeled graphs that represent our model fragments. Now a matching link is described by a pair composed of a source fragment (whose corresponding graph is in the extent of some concepts in the source graph lattice) and a target fragment (whose corresponding graph is in the extent of some concepts in the source graph lattice). This is described in a formal context, where objects are the matching links and attributes are the concepts of the two lattices (source graph lattice and target graph lattice). In this formal context (presented in Table 11(a)), a matching link is associated with the concepts having respectively its source graph and its target graph in their extent. This means that the matching link is described by the graph of its source fragment and by the generalizations of this graph in the lattice. This is the same for the graphs of the target fragments. For example, matching link `L0`, connecting source fragment 0 to target fragment 0, is associated in the formal context to concepts `sfc01`, `sfc012`, `tfc01`, `tfc012`.



(a) Target graph 1

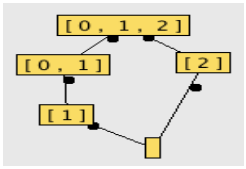


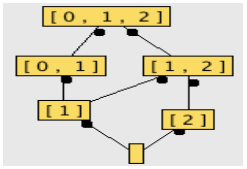
(b) Target graph 0



(c) Target graph 2

Fig. 7. Target graphs





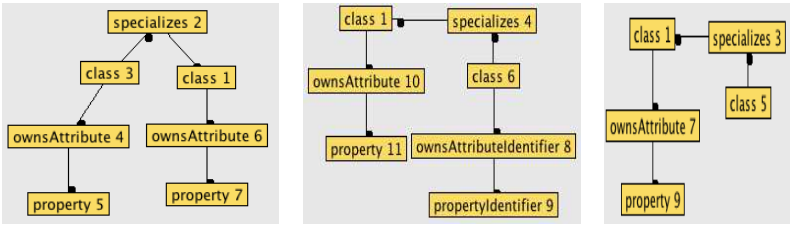
(a) Source graph lattice (b) Target graph lattice

Fig. 8. Graph lattices. Only concept extents are represented in the figure. Intents of concepts are shown in Fig. 9 and 10. We denote by $sfcx_1...x_n$ (resp. $tfcx_1...x_n$) the vertex [x_1,...,x_n] of the source (resp. target) graph lattice.

The concept lattice associated with the matching link formal context of Fig. 11(a) is shown in Fig. 11(b). In this representation (obtained with RCAexplore[1]) each box describes a concept: the first compartment informs about the name of the concept, the second shows the simplified intent (here concepts from source fragment lattice and target fragment lattice) and the third one shows the simplified extent (here matching links). `Concept_MatchingLinksFca_4` extent is composed of the links `L0` and `L1`, while the intent is composed of source graph concepts `sfc01, sfc012` and target graph concepts `tfc01, tfc012`.

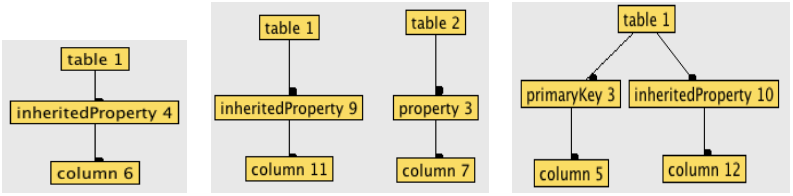
Model transformation pattern mining The last step of the process consists in extracting model transformation patterns from the matching link lattice. This has close connections to the problem of extracting implication rules in a concept lattice, but using only pairs of source and target graph concepts. The more

[1] http://dolques.free.fr/rcaexplore.php



(a) Concept `sfc2` (b) Concept `sfc01` (c) Concept `sfc012`

Fig. 9. Source graph lattice concepts. Concept `sfc1` (not represented) has Source Graph 1 of Fig. 6 as intent



(a) Concept `tfc012` (b) Concept `tfc12` (c) Concept `tfc01`

Fig. 10. Target graph lattice concepts. Concepts `tfc1` and `tfc2` (not represented) have resp. Target Graph 1 and Target Graph 2 from Fig. 7 as their intents.

reliable transformation patterns are given when using a source graph and a target graph in the same simplified intent of a concept, because this corresponds to the fact that the source graph is always present when the target graph is present too (and reversely). For example, from `Concept_MatchingLinksFca_0`, we obtain the following transformation pattern:

$$\textit{graph of } \texttt{sfc012} \textit{ intent} \hookrightarrow \textit{graph of } \texttt{tfc012} \textit{ intent}$$

This pattern expresses a new transformation pattern (*new* in the sense that it does not directly come from a matching link):

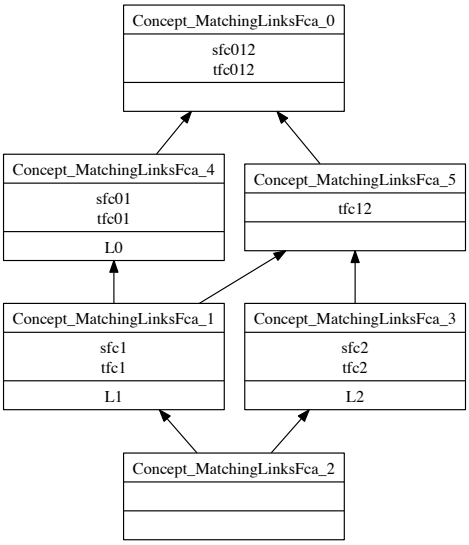
> *A UML model where a class C_d specializes another class C_m which owns an attribute a is transformed into a relational model where a table T owns a (inherited) column c.*

Due to the simplicity of our illustrative example, the other reliable patterns obtained from source and target graphs from the same simplified intent just correspond to matching links.

Obtaining other, less reliable patterns, relies on the fact that if a source graph and a target graph are not in the same simplified intent, but the concept

ML	sfc1	sfc2	sfc01	sfc012	tfc1	tfc2	tfc01	tfc12	tfc012
L0			×	×			×		×
L1	×		×	×	×		×	×	×
L2		×		×		×		×	×

(a) Matching Link Formal Context $MLFC$



(b) Matching Link Lattice

Fig. 11. Matching link formal context and corresponding concept lattice.

C_s which introduces the source graph is below the concept C_t which introduces the target graph, then we infer the following transformation pattern:

$$part\ of\ graph\ of\ C_s\ intent \hookrightarrow graph\ of\ C_t\ intent$$

For example, as `sfc1` appears below `tfc12`, we can deduce that, when the input of the transformation contains the graph of intent of`sfc1`, thus the output contains the graph of intent of `tfc12`. These patterns are less reliable, because the source graph may contain many things that have nothing to do with the target graph (compare `sfc1` and `tfc12` to see this phenomenon). However, experts can have a look on these patterns to find several (concurrent) transformation patterns when several source model fragments are transformed into a same target model fragment. We have a symmetric situation when a source graph and a target graph are not in the same simplified intent, but the concept C_t which introduces the target graph is below the concept C_s which introduces the source graph.

4 Feasibility study

We evaluated the feasibility of the approach on two different realistic transformation examples: (1) UML class diagram to relational schema model that contains

108 model elements, 10 fragments (5 sources, 5 targets) and 5 matching links (U2S) and (2) UML class diagram to entity relationship model that contains 66 model elements, 6 fragments (3 sources, 3 targets) and 3 matching links (U2E). We compute from the obtained graphs for each transformation example several pattern categories (see left-hand side of Table 1).

(1) The transformation patterns coming from simplified intents (which we think are the most relevant patterns): they correspond to graphs pairs (G_S, G_T) such that G_S and G_T are in the simplified intent of a same concept. They can be divided into two sets. The set TP_l groups the patterns that are inferred from the initial matching links (G_S, G_T are the ends of a matching link). The set TP_n contains the patterns that are learned from graph generalization and matching link classification.

(2) The transformation patterns ($TP_{n_{part_s}}$) coming from the graphs G_S and G_T, such that G_S is in simplified intent of a concept C_s which is a subconcept of the concept C_t which has G_T in its simplified intent and all concepts greater than C_s and lower than C_t have an empty simplified intent. In addition, we consider only the case where simplified intent of C_s contains only source graphs or (inclusively) simplified intent of C_t contains only target graphs.

(3) Symmetrically, the transformation patterns ($TP_{n_{part_t}}$) coming from the graphs G_S and G_T, such that G_T is in simplified intent of a concept C_t which is a subconcept of the concept C_s which has G_S in its simplified intent and all concepts greater than C_t and lower than C_s have an empty simplified intent. In addition, we consider only the case where simplified intent of C_s contains only source graphs or (inclusively) simplified intent of C_t contains only target graphs.

Table 1. Results. Left-hand side: sets cardinals. Right-hand side: precision metrics

	$\#TP_l$	$\#TP_n$	$\#TP_{n_{part_s}}$	$\#TP_{n_{part_t}}$
Ill. ex.	2	2	2	1
U2S	1	5	3	0
U2E	2	2	1	1

	P_{TP_l}	P_{TP_n}	$P_{TP_{n_{part_s}}}$	$P_{TP_{n_{part_t}}}$
Ill. ex.	1	1	0.72	0.72
U2S	1	0.75	0.78	-
U2E	1	1	0.73	0.95

We also evaluate each extracted transformation pattern using a precision metric. Precision here is the number of elements in the source and target graphs that participate correctly to the transformation (according to a human expert) divided by the number of elements in the graphs. We then associate a precision measure to a set of transformation patterns, which is the average of the precisions of its elements (See right-hand side of Table 1).

The results show that we learn transformation patterns that correspond to the initial mapping links. These patterns are relevant and efficient (precision = 1). 17 new transformation patterns are also learned from the three used examples. These patterns seems also relevant, with a precision average than 0.83.

5 Related Work

Several approaches have been proposed to mine model transformation. The MTBE approach consists in learning model transformation from examples. An example is composed of a source model, the corresponding transformed model, and matching links between the two models. In [1,15], an alignment between source and target models is manually created to derive transformation rules. The approach of [5] consists in using the analogy to search for each source model its corresponding target model without generating rules. In a previous work [12], we use Relational Concept Analysis (RCA) to derive commonalities between the source and target meta-models, models and transformation links to learn executable transformation rules. The approach based on RCA builds transformation patterns that indicate how a model element, in a specific context, is transformed into a target element with its own specific context. This approach has many advantages for the case when the matching link type is one-to-one, but it is not able to capture the cases where a set of model elements is globally transformed into another set of model elements (matching link type is many-to-many). In this paper, we investigate graph mining approaches, to go beyond the limitations of our previous work. In the current context of MDE, transformation examples are not very large (they are manually designed), thus we do not expect scalability problems. Compared with a solution where we would build a lattice on graphs containing elements from both source and target models coming from matching links, the solution we choose separately classifies source graphs and target graphs. This is because source graphs and target graphs could come from the same meta-model (or from meta-models with common concepts) and it has no meaning in our context to generalize a source graph and a target graph together. We also think that the result is more readable, even in the case of disjoint meta-models.

Our problem has close connections with the pattern structure approach [4] when the pattern structure is given by sets of graphs that have labeled vertices. Graph mining approaches [2,10] aim at extracting repeated subgraphs in a set of graphs. They use a partial order on graphs which usually relies on morphism or on injective morphism, also known as subgraph isomorphism [9]. In the general case, these two morphism operations have an exponential complexity. In this paper, we rely on graph mining to classify independently the origins and the destinations of matching links and to infer from this, a classification of matching links, that is then used to extract transformation patterns.

6 Conclusion

We have proposed an approach to assist a designer in her/his task of writing a declarative model transformation. The approach relies on model transformation examples composed of source and target model fragments and matching links. Models and their fragments are represented by graphs with labelled vertices that are classified. This classification is in turn, used for classifying the matching links.

Finally, the mined model transformation patterns express how a source model fragment is transformed into a target model fragment. Future directions of this work include extending the evaluation to other kinds of source and target metamodels, and define a notion of support for the patterns. We also would like to explore the different kinds of graph mining approaches, in particular to go beyond the limitation of using locally injective graphs. Finally, we plan to apply our approach [12] to transform the obtained patterns into operational rules.

References

1. Balogh, Z., Varro, D.: Model Transformation by Example Using Inductive Logic Programming. Software and Systems Modeling 8(3), 347–364 (2009)
2. Cook, D.J., Holder, L.B.: Mining Graph Data. John Wiley & Sons (2006)
3. Fabro, M.D.D., Valduriez, P.: Towards the efficient development of model transformations using model weaving and matching transformations. Software and System Modeling 8(3), 305–324 (2009)
4. Ganter, B., Kuznetsov, S.O.: Pattern structures and their projections. In: Proc. of ICCS'01. pp. 129–142 (2001)
5. Kessentini, M., Sahraoui, H., Boukadoum, M.: Model transformation as an optimization problem. In: MODELS'08, LNCS 5301. pp. 159–173. Springer (2008)
6. Lieberman, H. (ed.): Your Wish Is My Command: Programming by Example. Morgan Kaufmann Publishers (2001)
7. Liquiere, M., Sallantin, J.: Structural machine learning with Galois lattice and Graphs. In: Proc. of ICML'98. pp. 305–313 (1998)
8. Lopes, D., Hammoudi, S., Abdelouahab, Z.: Schema matching in the context of model driven engineering: From theory to practice. In: Advances in Systems, Computing Sciences and Software Engineering. pp. 219–227. Springer (2006)
9. Mugnier, M.L.: On generalization/specialization for conceptual graphs. J. Exp. Theor. Artif. Intell. 7(3), 325–344 (1995)
10. Nijssen, S., Kok, J.N.: The Gaston Tool for Frequent Subgraph Mining. Electr. Notes Theor. Comput. Sci. 127(1), 77–87 (2005)
11. Norris, E.: An algorithm for computing the maximal rectangles in a binary relation. Revue Roumaine Math. Pures et Appl. XXIII(2), 243–250 (1978)
12. Saada, H., Dolques, X., Huchard, M., Nebut, C., Sahraoui, H.A.: Generation of operational transformation rules from examples of model transformations. In: MoDELS. pp. 546–561 (2012)
13. Saada, H., Huchard, M., Nebut, C., Sahraoui, H.A.: Model matching for model transformation - a meta-heuristic approach. In: Proc. of MODELSWARD. pp. 174–181 (2014)
14. Weichsel, P.M.: The Kronecker product of graphs. Proceedings of the American Mathematical Society 13(1), 47–52 (1962)
15. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: Proc. of HICSS '07. p. 285b (2007)